User-guided Repairing of Inconsistent Knowledge Bases

Abdallah Arioua, Angela Bonifati

User-guided Repairing of Inconsistent Knowledge Bases

Abdallah Arioua*
University Lyon 1 & CNRS Liris
abdallah.arioua@univ-lyon1.fr

Angela Bonifati†
University Lyon 1 & CNRS Liris
angela.bonifati@univ-lyon1.fr

ABSTRACT

Repairing techniques for relational databases have leveraged integrity constraints to detect and then resolve errors in the data. User guidance has started to be employed in this setting to avoid a prohibitory exploration of the search space of solutions. In this paper, we present a user-guided repairing technique for Knowledge Bases (KBs) enabling updates suggested by the users to resolve errors. KBs exhibit more expressive constraints with respect to relational tables, such as tuple-generating dependencies (TGDs) and negative rules (a form of denial constraints). We consider TGDs and a notable subset of denial constraints, named contradiction detecting dependencies (CDDs). We propose user-guided polynomial-delay algorithms that ensure the repairing of the KB in the extreme cases of interaction among these two classes of constraints. To the best of our knowledge, such interaction is so far unexplored even in repairing methods for relational data. We prove the correctness of our algorithms and study their feasibility in practical settings. We conduct an extensive experimental study on synthetically generated KBs and a real-world inconsistent KB equipped with TGDs and CDDs. We show the practicality of our proposed interactive strategies by measuring the actual delay time and the number of questions required in our interactive framework.

1 INTRODUCTION

Integrity constraints have been used in relational databases to detect inconsistencies and thus repair error-prone data within tables. Notable classes of these constraints are represented by functional dependencies (FDs) and conditional functional dependencies (CFDs) that are both table-level constraints as they express conditions without or with predicates on entire relations. Denial constraints (DCs) [7] are more general first-order formulas that encompass FDs and CFDs and strike a balance between expressiveness and complexity. Denial constraints are, however, difficult to understand for end users and their intractability and prohibitive search space make them unattractive for repairing algorithms [3].

In this paper, we focus on a subset of denial constraints, which we call contradiction-detecting dependencies (CDDs) capturing contradictions in the data. CDDs correspond to denial constraints restricted to equality predicates in their respective bodies. They are used mainly to capture contradictions and disjointness between relations. They differ from other subfamilies of DCs such as keys, functional dependencies and equality-generating dependencies. Knowledge Bases (KBs) typically rely on the interaction of CDDs (also known as negative rules or negative constraints) with tuple-generating dependencies (also called existential rules) [5, 11]. The following example underlines the importance of CDDs.

*Supported by PALSE Impulsion.
†Partially supported by CNRS Mastodons MedClean and PALSE Impulsion.

Example 1.1. Figure 1 (a) shows our running example. Let $\mathcal{F}$ contain the set of facts of a KB describing the prescriptions of patients at a hospital and Σ_C the set of CDDs. *Aspirin* is prescribed to *John* who is allergic to it, whereas *Mike* has an allergy against *Penicillin*. The CDD in Σ_C dictates that prescribing a drug to a person who has an allergy against it leads to a contradiction.

Several approaches to repair KBs exist, such as deletion-based repairing, which amounts to remove the inconsistencies in order to satisfy the constraints. However, the generated repairs are not compatible, as a consequence, choosing among them is not feasible for end-users, as shown by the following example.

Example 1.2. Following the deletion-based repairing approaches, one can either remove *prescribed(Aspirin,John)* or *hasAllergy(John, Aspirin)* as either one of them is false according to the CDD. The first one gives us the repair $\mathcal{F}_1$ that conveys the information that *Aspirin* is prescribed to *John*. Conversely, the second repair $\mathcal{F}_2$ would lead us to conclude that *John* has an allergy against *Aspirin*. Moreover, none of the above repairs preserves as much information as possible. The information about *John* having an allergy that could be against *Aspirin* or any other drugs is indeed lost in $\mathcal{F}_1$ whereas the information that *Aspirin* is prescribed to someone which could be *John* or someone else is also lost in $\mathcal{F}_2$. This example also shows the impossibility for an end-user of making a choice between these two repairs.

An alternative to deletion-based repairing is given by update-based repairing [24, 28], which inspired our work. In update-based repairing, atomic values can be modified instead of removing entire facts from the knowledge base.

Example 1.3. By applying an update-based repairing to the above inconsistent knowledge base, we obtain the set of facts $\mathcal{F}_3$ (in which X_1 is a labeled null referring to an unknown allergy). Another possible repair is that *John* has an allergy against *Penicillin* rather than *Aspirin*. Or, *Penicillin* is prescribed to *John* rather than *Aspirin*. Clearly, all these update-based repairs preserve more facts than the deletion-based ones illustrated above.

Although being beneficial, update-based repairing suffers from the problem of choosing the positions to modify and the value to use in repairing. For instance, do we need to change *Aspirin* to *Penicillin* in *hasAllergy*(*John*, *Aspirin*) or *Aspirin* to a labeled null? Clearly, user intervention is compulsory in such a case in order to reach a repair that meets the user's requirements and his expertise about the domain.

The problem becomes more complex when CDDs and TGDs are considered as shown in Figure 1 (b). Besides the fact that $\mathcal{F}$ already contains inconsistencies as illustrated in Figure 1 (a), we consider Σ_T and the new introduced atoms in $\mathcal{F}$ and a new CDD in $\Sigma_C{}'$. In this KB, another inconsistency is raised due to the interaction between TGDs and CDDs corresponding to the fact that *John* was prescribed incompatible drugs, i.e. *Aspirin* and *Nsaids*. Such a contradiction can only be discovered after applying the TGD that results in deducing the fact *John* is prescribed *Nsaids*, because he has *Migraine* pain and *Nsaid* is a painkiller. In such a

$\mathcal{F} = \{prescribed(Aspirin, John), hasAllergy(John, Aspirin), hasAllergy(Mike, Penicillin)\}$
$\Sigma_C = \{prescribed(X, Y), hasAllergy(Y, X) \rightarrow \bot\}$
$\mathcal{F}_1 = \{prescribed(Aspirin, John), hasAllergy(Mike, Penicillin)\}$
$\mathcal{F}_2 = \{hasAllergy(John, Aspirin), hasAllergy(Mike, Penicillin)\}$
$\mathcal{F}_3 = \{prescribed(Aspirin, John), hasAllergy(John, X_1), hasAllergy(Mike, Penicillin)\}$

(a) An inconsistent knowledge base with only CDDs. $\mathcal{F}_i$ are repairs.

$\mathcal{F}' = \mathcal{F} \cup \{hasPain(John, Migraine), isPainKillerFor(Nsaids, Migraine), incompatible(Aspirin, Nsaids)\}$
$\Sigma_T = \{isPainKillerFor(X, Y), hasPain(Z, Y) \rightarrow prescribed(X, Z)\}$
$\Sigma_C' = \Sigma_C \cup \{prescribed(X, Z), prescribed(X, Y), incompatible(Y, Z) \rightarrow \bot\}$

(b) An inconsistent knowledge base with CDDs and TGDs.

Figure 1: Examples on tuple-based repairing and update-based repairing.

case, after applying the rule in Σ_T, a new inconsistency has been introduced. Hence, the choice of which inconsistency to handle first and which atom to update is crucial. For instance, updating the atom $prescribed(Aspirin, John)$ will resolve automatically the new inconsistency without updating other atoms, whereas updating the atom $prescribed(Nsaids, John)$ will not. In addition, propagating back the changed positions in $prescribed(Nsaids, John)$ should be done in order to establish consistency.

In this paper, we present a user-guided update-based repairing framework that is capable of solving contradictions triggered by CDDs. We study the interaction of such rules with more classical tuple-generating dependencies (or, existential rules) in KB reasoning. The study of DCs in a relational setting has been extensively conducted in the literature as witnessed by several papers in the area [13, 24, 28]. We defer the discussion of the differences between our work and previous work to the next subsection. In this paper, we focus on the following problem statement, which substantially deviates from the objectives of previous work.

(URP) Given a KB equipped with a set of TGDs and CDDs, the User-guided Repairing Problem *is to compute, by means of user's update fixes, an error-free KB by addressing two main challenges: (i) minimizing user interactions and (ii) accounting for the interplay of TGDs and CDDs. If the user is an oracle, then the repair of the KB is also a u-repair, i.e. a repair with a minimal (w.r.t $\subseteq$) set of update fixes.*

Contributions. The main contributions of our paper are as follows:

(1) Update-based repairing: We introduce contradiction detection dependencies (CDDs) and we formalize update-based repairing in the presence of both CDDs and TGDs. We prove that repairability is guaranteed and can be checked in polynomial time.

(2) Update-based repairing with user interaction: We define an interactive framework letting the user repair the knowledge base and meet his requirements. We prove two interesting properties: *(i) soundness*, i.e. we show that the framework is sound, which means that for every dialogue with a user we can reach a consistent state of the knowledge base; *(ii) soundness w.r.t. an oracle*, i.e. we assume that the interaction is done with an oracle that has a specific repair in mind and we prove that the output of the dialogue with an oracle is exactly the repair of the oracle. We show that the dialogue algorithm has a polynomial delay in generating questions.[1]

We present an extensive experimental study, devoted to confirm the polynomiality of delay time and showing the feasibility of our interactive approach in terms of number of questions and average number of conflicts per question in the knowledge bases. In our assessment, we contrast the two cases of only CDDs and CDDs alongside with TGDs and we study the impact of the chase algorithm on the proposed interactive strategies in both cases.

Paper organization. The paper is organized as follows. In Section 2, we introduce the basic notions and definitions. In Section 3, we introduce the update-based repairing framework and the repairability of KBs. In Section 4, we formalize user intervention by means of the notion of inquiry and we prove the soundness and termination of an inquiry engaging the end-user. We also discuss the case in which the user is an oracle and prove that the inquiry has a polynomial delay time. In Section 5, we introduce different interactive strategies with the user. In Section 6, we present our experimental assessment. Finally, Section 7 concludes the paper.

1.1 Related Work

Rule-Based Repairing. Logical data cleaning has leveraged reasoning over more or less sophisticated classes of declarative dependencies [9, 14] in order to detect and repair error-prone values and tuples. A plethora of data quality constraints have been introduced to this purpose, ranging from classical functional dependencies and their approximate variants for relational tables [9, 27] to their counterparts in graph databases [17]. Denial constraints [7] are first-order formulas more expressive than functional dependencies and conditional functional dependencies. Comprehensive classes of graph constraints, including the expressive graph entity dependencies (encompassing denial constraints) have been presented in [16]. The detection problem [17] for these constraints consists in checking whether a given database (or a graph) contains no violations of the input set of constraints. While [16] focuses on the detection problem along with satisfiability and implication among constraints, our goal in this paper is to compute an update-based repair for a knowledge base with an interactive exploration of the search space of solutions. Our contradiction detecting dependencies are a subset of denial constraints, limited to equality as built-in predicate. While denial constraints have been already employed as data quality rules in the relational setting [13], their use in the realm of knowledge bases characterized by the schema-less nature of data and their combination with other KB constraints, such as TGDs, is not explored. CLAMS [18] investigates the use of DCs in data lakes, including RDF KBs, but it does not consider the TGDs and their interactions with DCs. With that being said, our approach goes with the same line of [13, 18] in reinforcing and endorsing a holistic view of repairing for knowledge bases by compiling the information coming from multiple violations in a structure called the Conflicts Hypergraph [13, 24]. Repairing a database [1, 7, 28] according to a set of constraints corresponds to bringing the database to its legal state, in which all the constraints are satisfied. Since many possible repairs for a given database may exist, one would tend to prefer the (minimal) one, which entails less modifications of the original database. Various notions

[1]The delay between the asked questions is bounded by a polynomial.

of repairs have been used and many approaches have used insert and delete operations on the original database to make it reach its consistent state with respect to a given class of constraints. Such approaches may exhibit drawbacks in that the granularity of the operations performing the repairs is too coarse. Indeed, deletions and insertions are typically executed at the tuple level for relations, thus leading to discard possibly error-free values. Our work has been inspired by update-based repairing proposed in [10, 28] to allow value replacement on positional attributes in relational tuples. While [28] focused on consistent query answering for update repairs aiming at finding the answers of a query in the intersection of all possible repairs, our intent is to exploit user interactions in the update-based repairing process of an entire knowledge base. Repairing by value modification with functional dependencies and inclusion dependencies has been tackled in [10] with the aim of building minimal-cost repairs. Their algorithms are not directly applicable to KBs due the inherent difference of expressiveness of the constraints and the consequent interaction between tuple-generating dependencies and the CDDs. In addition, user intervention has not been considered in [10].

User-guided data cleaning. A fruitful line of work has led to the design of several data cleaning tools, such as Llunatic [19], GDR [29], Katara [14], Dance [4] and Falcon [21].

GDR [29] considers user-guided relational data repairing. CFDs are used to generate candidate updates for the tuples that are violating them. The user is presented with groups of updates and her feedback is fed into an active learning process that decides about the correctness of updates without user involvement. The convergence of updates in our method is ensured by the chase algorithm involving CDDs and TGDs on KBs.

Llunatic [19] is mapping and cleaning tool accepting user suggestions during the chase procedure with EGDs on relational instances. Llunatic also explores the interaction among several classes of constraints such as FDs, CFDs, editing rules and TGDs. To the best of our knowledge Llunatic cannot be directly applied to knowledge bases with constraints such as CDDs and TGDs.[2]

Falcon [21] relies on a set of SQL update queries instead of a set of input logical constraints to entail the repair of a relational database. A set of SQLU queries is inferred starting from one triggering input tuple-based update proposed by the non-expert user. Our approach is based on rule-based repairing of knowledge bases and on a tight interaction with the domain expert to perform data curation, not considered in the above system.

Dance [4] introduces a user-driven cleaning approach for relational tuples, by considering constraints similar to classical EGDs and TGDs. Dance proposes a set of suspicious tuples whose update can contribute to constraint resolution. However, neither they consider DCs or subset thereof employed in our framework nor they leverage their interaction with TGDs as in our approach.

Katara [14] is orthogonal to our work in that it leverages knowledge bases and guidance from crowdsourcing to fix the errors in RDBMS. Because of that, input KBs are assumed to be well curated, as opposed to the assumption undertaken in our paper.

2 PRELIMINARIES

In this section, we briefly recap the notions needed in our framework, namely the definition of a knowledge base and the corresponding constraints, along with the definition of conflicts.

[2]Benchmarks on LUBM in [6] have been performed on a relational representation of the LUBM ontology with vertical partitioning.

Constraints and KBs. A *tuple-generating dependency* (abbreviated TGD) is of the form $R : \forall\mathbf{x}\forall\mathbf{y}B(\mathbf{x},\mathbf{y}) \rightarrow \exists\mathbf{z}H(\mathbf{y},\mathbf{z})$, where $\mathbf{x}$ and $\mathbf{y}$ are sequences of variables, B and H are conjunctions of atoms, with $vars(B) = \mathbf{x}\cup\mathbf{y}$, and $vars(H) = \mathbf{y}\cup\mathbf{z}$. B and H are respectively called the *body* and the *head* of R. A *contradiction-detecting dependency* (abbreviated CDD) is of the form $N : \forall\mathbf{x}B(\mathbf{x}) \rightarrow \bot$ where B is a conjunction of atoms, with $vars(B) = \mathbf{x}$. The body B may have equalities but no inequalities [12]. Inequalities are not used as they lead to undecidability even for TGDs [20]. Notice that whereas CDDs are a subset of DCs (Denial Constraints), they are different from Keys, FDs and EGDs (subsets of DCs).

A dependency with an empty body B and a non-empty head H is called a *fact*. Therefore, a fact is a set of atoms with existential variables (i.e. labeled nulls). A knowledge base $\mathcal{K} = (\mathcal{F}, \Sigma_T, \Sigma_C)$ consists of a finite sets of facts, TGDs and CDDs, respectively. Reasoning with a knowledge base is done via the chase. A rule $R : B \rightarrow H$ is *applicable* to a fact F if there exists a homomorphism π from B to F. The *application of R to F w.r.t. π* produces a finite set of atoms (also called atomset) $\alpha(F, R, \pi) = F\cup\pi(safe(H))$, where $safe(H)$ is obtained from H by replacing existential variables with fresh variables. The application of all TGDs to a set of facts is called the chase. The result of the chase on $\mathcal{F}$ is denoted as $C\ell_{\Sigma_T}(\mathcal{F})$ which produces an expanded set of facts $\mathcal{F}^*$. In this paper, we restrict ourselves to weakly-acyclic TGDs to avoid non-terminating chase sequences [15]. Let us consider the example in Figure 1 (b), on which we show the result of the chase.

Example 2.1. The result of the chase on the set of fact $\mathcal{F}'$ is: $C\ell_{\Sigma_T}(\mathcal{F}') = \mathcal{F}' \cup \{prescribed(Nsaids, John)\}$.

Query Answering. Given a set of facts $\mathcal{F}$, an answer to Q in $\mathcal{K} = (\mathcal{F}, \Sigma_T, \Sigma_C)$ is a tuple of constants $(A_1, \ldots, A_k)$ such that there exists a homomorphism π from Q to $C\ell_{\Sigma_T}(\mathcal{F})$, with $(A_1, \ldots, A_k) = (\pi(x_1), \ldots, \pi(x_k))$. We denote by $Q(\mathcal{F}, \Sigma_T)$ the set of all answers of Q over $\mathcal{F}$ in presence of Σ_T.

Inconsistent knowledge bases and conflicts. A widely accepted assumption in KBs is that the set of TGDs is compatible with the set of CDDs, i.e. the union of the two sets is satisfiable [25]. A set of facts $\mathcal{F}$ is inconsistent with respect to a set of TGDs Σ_T and CDDs Σ_C (or inconsistent for short) if and only if there exists a dependency $N \in \Sigma_C$ such that $C\ell_{\Sigma_T}(\mathcal{F}) \models body(N)$. A knowledge base $\mathcal{K} = (\mathcal{F}, \Sigma_T, \Sigma_C)$ is inconsistent if and only if there exists a set of facts $\mathcal{F}' \subseteq \mathcal{F}$ such that $\mathcal{F}'$ is inconsistent. We use the alternative notation $C\ell_{\Sigma_T}(\mathcal{F}) \models \bot$ hereafter.

Example 2.2 (Example 2.1 Ct'd). The knowledge base $\mathcal{K} = (\mathcal{F}', \Sigma_T, \Sigma_C{}')$ is inconsistent because the bodies of the two CDDs are entailed from $C\ell_{\Sigma_T}(\mathcal{F}')$.

Inconsistency can also be characterized by conflicts.

Definition 2.3 (Conflict). Let $\mathcal{K} = (\mathcal{F}, \Sigma_T, \Sigma_C)$ be an inconsistent knowledge base. A conflict is defined as a tuple $\mathcal{X}$=(N, h) such that h is a homomorphism from $body(N)$ to $C\ell_{\Sigma_T}(\mathcal{F})$ such that $h(body(N)) \subseteq C\ell_{\Sigma_T}(\mathcal{F})$.

Example 2.4. The knowledge base $\mathcal{K} = (\mathcal{F}', \Sigma_T, \Sigma_C{}')$ has two conflicts $\mathcal{X}_1 = (N_1, h_1)$ and $\mathcal{X}_2 = (N_2, h_2)$ defined as follows:

- $N_1 = prescribed(X, Y), hasAllergy(Y, X) \rightarrow \bot$.
- $h_1(X) = Aspirin$, $h_1(Y) = John$
- $N_2 = prescribed(X, Z), prescribed(Y, Z),$ $incompatible(X, Y) \rightarrow \bot$.
- $h_2(X) = Aspirin$, $h_2(Y) = Nsaids$.

We denote by $\texttt{conflict}(\mathcal{K},N)$ all the conflicts for a given constraint $N \in \Sigma_C$. The set of all conflicts of a given knowledge base is denoted as:

$$\texttt{allconflicts}(\mathcal{K}) = \bigcup_{N \in \Sigma_C} \texttt{conflict}(\mathcal{K}, N)$$

A knowledge base $\mathcal{K}$ is consistent iff $\texttt{allconflicts}(\mathcal{K}) = \emptyset$.

In order for CDDs to be meaningful, we impose that CDDs contain atoms with join variables. This assumption is made to avoid for instance CDDs of the form $prescribed(X, Y) \rightarrow \bot$ in the above example. Such CDD is a schema constraint imposing that *prescribed* should be removed from the vocabulary of the KB.

3 UPDATE-BASED REPAIRING

In this section, we introduce the framework of update-based repairing for KBs. As opposed to deletion-based repairing, the granularity of update-based repairing is no longer an atom but instead a position that we need to update within a given atom. In what follows, we introduce the concept of a position and a fix on a position. Then, we proceed by giving the definition of a repair in such context, i.e. based on a minimal set of fixes needed to be applied in order to recover the consistency of a KB.

Given an atom $A = p(t_1, \ldots, t_n)$, we denote by $\texttt{arity}(A) = n$ the arity of the predicate $\texttt{pred}(A) = p$. The tuple (A, i) such that $i \in [1, \texttt{arity}(A)]$ is called a position and *identifies* the position of the i-th argument of p. We denote by $\texttt{adom}(A, i, \mathcal{F})$ the active domain of the argument i of p in $\mathcal{F}$.

A position is a building block in update-based repairing as it gives access to the inner structure of an atom. For instance, $(A, 1)$ such that $A = prescribed(Aspirin, John)$ is a position that refers to the first argument of A.

Given $\mathcal{F}$, the set of all positions of $\mathcal{F}$ is defined as:

$$\texttt{pos}(\mathcal{F}) = \{(A, i) \mid A \in \mathcal{F} \textit{ and } i \in [1, \texttt{arity}(A)]\}$$

The function $\texttt{value}^i_A(\mathcal{F})$ returns the value of the position (A, i). Since existential variables can be present in atoms, $\texttt{value}^i_A(\mathcal{F})$ can be either an existentially quantified variable or a constant. The set of all values of a set of facts $\mathcal{F}$ is defined as:

$$\texttt{vals}(\mathcal{F}) = \{\texttt{value}^i_A(\mathcal{F}) \mid (A, i) \in \texttt{pos}(\mathcal{F})\}.$$

A position fix specifies an update on a given atom in a given position.

Definition 3.1 (Position fix). A *fix* on a position (A, i) in $\mathcal{F}$ is a triple (A, i, t) such that $t \in \texttt{adom}(A, i, \mathcal{F}) \setminus \texttt{value}^i_A(\mathcal{F})$ or $t = X^j_A$ is an existential variable that is uniquely attributed to (A, i).

A fix on a position can specify a value that is within the active domain of the predicate p and different from the actual value. A fix can also specify an existential variable that refers to an unknown individual. Please note that such a variable is unique to the position in question and it is not used elsewhere in the knowledge base.

The *application* of a set of fixes $\mathcal{P}$ on $\mathcal{F}$ is defined as follows:

$$\texttt{apply}(\mathcal{F}, \mathcal{P}) = \{p(t'_1, \ldots, t'_n) \mid A = p(t_1, \ldots, t_n) \in \mathcal{F} \textit{ and } \forall i \in \{1, \ldots, n\} \textit{ either } (A, i, t'_i) \in \mathcal{P} \textit{ or } (A, i, t'_i) \notin \mathcal{P} \textit{ and } t'_i = t_i\}$$

We consider only *valid* set of fixes which are set of fixes $\mathcal{P}$ such that there exist no two fixes $(A, i, t), (A, i, t') \in \mathcal{P}$ and $t \neq t'$. The application of a set of fixes $\mathcal{P}$ on a set of facts gives another set of facts called *the update* of $\mathcal{F}$ by $\mathcal{P}$. It is clear that $|\mathcal{F}'| = |\mathcal{F}|$ and $\texttt{pos}(\mathcal{F}') = \texttt{pos}(\mathcal{F})$.

Example 3.2. The following is a set of fixes $\mathcal{P} = \{(A, 2, X_1), (A', 2, Aspirin)\}$ such that:

- $A = hasAllergy(John, Aspirin)$.
- $A' = hasAllergy(Mike, Penicillin)$.

The update of $\mathcal{F}$ by $\mathcal{P}$ gives:

- $\mathcal{F}'_1 = \{prescribed(Aspirin, John), hasAllergy(John, X_1), hasAllergy(Mike, Aspirin)\}$

The following set of fixes is not valid as it modifies the same position with different values:

- $\mathcal{P}' = \mathcal{P} \cup \{A, 2, Penicillin)\}$

An important notion that will be used later is the reconstruction of a set of fixes $\mathcal{P}$ given a set of facts $\mathcal{F}$ and its update $\mathcal{F}'$. We define the function $\texttt{diff}(\mathcal{F}, \mathcal{F}')$ as follows:

$$\texttt{diff}(\mathcal{F}, \mathcal{F}') = \{(A, i, t'_i) \mid A = p(t_1, \ldots, t_n) \in \mathcal{F}, A' = p(t'_1, \ldots, t'_n) \in \mathcal{F}' \textit{ and } match(A) = A' \textit{ and } \exists j \in \{1, \ldots, \texttt{arity}(A)\} \textit{ such that } t'_j \neq t_j\}$$

Notice that the function $match(x)$ puts the atoms of $\mathcal{F}$ and $\mathcal{F}'$ in one-to-one correspondence. Such one-to-one correspondence exists because we know that $\mathcal{F}'$ is an update of $\mathcal{F}$, therefore $|\mathcal{F}| = |\mathcal{F}'|$. $match(x)$ should satisfy the condition that $match(x) = y$ if and only if $x \in \mathcal{F}$ and $y \in \mathcal{F}'$ and $\texttt{pred}(x) = \texttt{pred}(y)$.

Example 3.3. Consider $\mathcal{F}$ of Example 1.1 and its update $\mathcal{F}'$ of Example 3.2, one can construct $\mathcal{P}$ by defining $match(A_1) = A'_1$, $match(A_2) = A'_2$ and $match(A_3) = A'_3$ such that:

- $A_1 = prescribed(Aspirin, John)$ and $A'_1 = prescribed(Aspirin, John)$.
- $A_2 = hasAllergy(John, Aspirin)$ and $A'_2 = hasAllergy(John, X_1)$.
- $A_3 = hasAllergy(Mike, Penicillin)$ and $A'_3 = hasAllergy(Mike, Aspirin)$.

Note that there may be finitely many one-to-one correspondences between two sets of facts.

The set of fixes $\mathcal{P}$ gives a consistent update $\mathcal{F}'$. In fact, it is minimal in the sense that only what is necessary to recover consistency has been changed. In what follows, we introduce the notion of consistent fixes, repair fixes and update repair.

Definition 3.4 (c-fix and r-fix). Let $\mathcal{K}$ be an inconsistent knowledge base, $\mathcal{P}$ a set of fixes and $\mathcal{F}' = \texttt{apply}(\mathcal{F}, \mathcal{P})$ the update of $\mathcal{F}$ by $\mathcal{P}$. $\mathcal{P}$ is called consistent fixes (denoted c-fix) of $\mathcal{K}$ iff $\mathcal{K} = (\mathcal{F}', \Sigma_T, \Sigma_C)$ is consistent. $\mathcal{P}$ is called repair fixes (denoted r-fix) of $\mathcal{K}$ iff $\mathcal{P}$ is a c-fix and it contains no c-fix $\mathcal{P}' \subset \mathcal{P}$.

$\mathcal{F}'$ is an update-repair if $\mathcal{P}$ is an r-fix. A c-fix is a set of fixes that gives a consistent update, an r-fix is a set of fixes that gives a consistent update that is minimal with respect to the changes.

Example 3.5. $\mathcal{P}$ is a c-fix and $\mathcal{P}_1 = \mathcal{P} \setminus \{(A', 2, Aspirin)\}$ is an r-fix. However, $\mathcal{P}_2 = \mathcal{P} \setminus \{(A, 2, X_1)\}$ is not a c-fix.

The following is a u-repair produced by $\mathcal{P}_1$:

$$\mathcal{F}_1 = \{prescribed(Aspirin, John), hasAllergy(John, X_1), hasAllergy(Mike, Penicillin)\}.$$

It is not hard to see that there exist finitely many r-fixes for a given set of facts $\mathcal{F}$ because a position can take a finite set of values assuming that the active domain is finite.

After having defined the basic notions for update-based repairing, in what follows we introduce Π-repairability, a key concept in our framework. For a given knowledge base $\mathcal{K}$, we are interested in knowing whether there always exists a way to repair $\mathcal{K}$. In Example 1.1, the knowledge base is repairable because there exists an r-fix for $\mathcal{F}$. In fact, for an arbitrary inconsistent knowledge base $\mathcal{K}$, repairability is guaranteed as one can change all positions to fresh existential variables, and since such variables are unique to the positions no constraint will be triggered. This gives us a c-fix,

consequently an r-fix for $\mathcal{K}$. Π-repairability is a generalization of repairability where Π refers to those positions that are *immutable* or not allowed to be changed. This generalization helps us to know whether the KB is repairable when some positions are modified by the user and not allowed to be changed.

Definition 3.6 (Π-repairability). Let $\mathcal{K}$ be an inconsistent knowledge base and $\Pi \subseteq \mathtt{pos}(\mathcal{F})$ be a set of positions. We say that $\mathcal{K}$ is Π-repairable if and only if there exists an r-fix $\mathcal{P}$ of $\mathcal{K}$ such that there exists no $(A, i, t) \in \mathcal{P}$ and $(A, i) \in \Pi$.

A knowledge base can be inconsistent but Π-repairable. In such case, Π-repairability indicates in a sense the possibility of finding a u-repair for $\mathcal{K}$ if certain positions are fixed prior to the repairing process. If $\mathcal{K}$ is not Π-repairable then $\mathcal{K}$ has no u-repair whose corresponding r-fix $\mathcal{P}$ changes the positions in $\mathtt{pos}(\mathcal{F}) \setminus \Pi$.

As stated above, Π-repairability is a generalization of the concept of repairability. When all positions are immutable then Π-repairability reduces down to a consistency check. Formally, if $\Pi = \mathtt{pos}(\mathcal{F})$ and $\mathcal{K}$ is Π-repairable then $\mathcal{K}$ is consistent.

Algorithm 1 for checking Π-repairability proceeds by changing all positions to fresh existential variables except those positions that belong to Π. Then, we check the consistency of this new knowledge base using CHECKCONSISTENCY($\mathcal{K}$). In fact, the algorithm checks if fixing some positions with their corresponding values will result in fixing the violations of some CDDs. If this is the case, the knowledge base can never be repaired.

Example 3.7. Consider the following knowledge base $\mathcal{K}$ with an empty Σ_T:

- $\mathcal{F} = \{p(a, b), q(b, d)\}$
- $\Sigma_C = \{p(X, Y), q(Y, Z) \rightarrow \bot\}$

If we take $\Pi = \emptyset$ then $\mathcal{K}$ is Π-repairable. This is because the c-fix $\mathcal{P} = \{(p(a, b), 1, X_1), (p(a, b), 2, X_2), (q(b, d), 1, X_3), (q(b, d), 1, X_4)\}$ gives a consistent update. Consequently, $\mathcal{P}$ is a c-fix. Necessarily, one can consider the r-fix $\mathcal{P}' = \{(p(a, b), 2, X_1)\} \subset \mathcal{P}$ which gives a u-repair. However, if $\Pi = \{(p(a, b), 2), (q(b, d), 1)\}$ then $\mathcal{K}$ is not Π-repairable because regardless of the values that the other positions can take the dependency will always be violated. Note that the fact that Σ_T is empty does not change the situation, given that the consistency check function is generic.

Checking Π-repairability is easy from a computational perspective. Algorithm 1 does perform such check in a polynomial time. The function CHECKCONSISTENCY($\mathcal{K}$) in Algorithm 1 evaluates on the body of every CDD $N \in \Sigma_C$ on $C\ell_{\Sigma_T}(\mathcal{F})$ and checks whether the query has an answer. If this is the case, $\mathcal{K}$ is inconsistent, otherwise it proceeds until no CDD is left to be evaluated, where the knowledge base achieves consistency. Clearly, the function Π-REP($\mathcal{K}$) runs in linear time of the size of $\mathtt{pos}(\mathcal{F})$ plus the computational overload of the function CHECKCONSISTENCY($\mathcal{K}$). This gives a polynomial data complexity as evaluating boolean conjunctive queries is polynomial in data complexity even in presence of weakly-acyclic TGDs [12, 22] .

We now need to prove that the algorithm is sound, i.e. if the knowledge base is Π-repairable then the algorithm produces true as an output, otherwise false.

PROPOSITION 3.8. *$\mathcal{K}$ is Π-repairable iff* Π-rep$(\mathcal{K}, \Pi) = true$.

PROOF. ($\Rightarrow$): suppose that $\mathcal{K}$ is Π-repairable and Π-REP($\mathcal{K}, \Pi$) returns false. The former implies that there exists an r-fix $\mathcal{P}'$ of $\mathcal{K}$ such that $\mathcal{F}'' = \mathtt{apply}(\mathcal{F}, \mathcal{P}')$ is the u-repair of $\mathcal{F}$ By $\mathcal{P}'$. The latter implies that $\mathcal{K}' = (\mathcal{F}', \Sigma_T, \Sigma_C)$ in line 5 is inconsistent, thus there exists a conflict $\mathcal{X} = (N, h)$ in $\mathcal{K}'$. Since there exists a homomorphism from $body(N)$ to $\mathcal{F}'$, we now show that $\mathcal{K}'' = (\mathcal{F}'', \Sigma_T, \Sigma_C)$ is necessarily inconsistent by constructing a homomorphism g from $\mathcal{F}'$ to $\mathcal{F}''$ i.e. $\mathcal{X}$ would also be a conflict in $\mathcal{K}''$, thus $\mathcal{K}''$ is inconsistent.

Recall that $\mathcal{P}'$ is the set of fixes that assigns to every position $(A, i) \in \mathtt{pos}(\mathcal{F}), (A, i) \notin \Pi$ a unique existential variable X^j_A. Let $\mathcal{P}'' = \mathtt{diff}(\mathcal{F}', \mathcal{F}'')$, we define the homomorphism $g : A \longmapsto B$ such that $A = \{X^i_A \mid (A, i, X^j_A) \in \mathcal{P}'\}$, $B = \{t \mid (A, i, t) \in \mathcal{P}''\}$, and $g(X^i_A) = t$ such that $(A, i, t) \in \mathcal{P}''$. Since there exists a homomorphism from $body(N)$ to $\mathcal{F}'$, and from $\mathcal{F}'$ to $\mathcal{F}''$ then there exists necessarily a homomorphism from $body(N)$ to $\mathcal{F}'$. Hence, $\mathcal{K}''$ is inconsistent.

($\Leftarrow$): it is trivial, if $\mathcal{K}'$ is consistent then $\mathcal{P}$ is a c-fix of $\mathcal{K}$ such that $\nexists (A, i, t) \in \mathcal{P}$, (A, i). By definition, $\exists \mathcal{P}' \subseteq \mathcal{P}$ such that $\mathcal{P}'$ is an r-fix of $\mathcal{K}$. □

Algorithm 1 Π-repairability

```
1: function Π-REP(K, Π)
2:     Π' ← pos(F) \ Π
3:     P ← {(A, i, t) | (A, i) ∈ Π' and t = X^i_A}
4:     F' ← apply(F, P)
5:     K' ← (F', Σ_T, Σ_C)
         return CHECKCONSISTENCY(K')
6: end function
```

We have introduced so far the key concepts of our framework. Nevertheless, as already mentioned in the introduction, update-based repairing is unfeasible in practice because there are no guidelines on (1) how to choose the positions among those possible, and (2) who provides the corresponding fixes. Our positioning here is that update-based repairing should go hand in hand with user intervention. In the next section, we introduce our interactive framework serving this purpose.

4 USER INTERVENTION

The key idea behind user intervention is that the user may have a repair in mind, which corresponds to how the knowledge base should turn to be consistent. Obviously, it is impossible for a user to manually repair the KB. In this section, we propose a framework of *inquiry dialogue* that takes a place between the knowledge base and the user. The basic idea is that the knowledge base asks questions about some fixes and the user chooses which one is true until he reaches a consistent knowledge base or, alternatively, a u-repair under some conditions.

Definition 4.1 (Inquiry). Given an inconsistent knowledge base $\mathcal{K}$ and a possibly empty set of positions Π. A **question** has the form $\phi = \{f_1, \ldots, f_n\}$ such that f_k is a fix. An **answer** to ϕ is a fix $f_k \in \phi$. Given a conflict $\mathcal{X} = (N, h)$ in $\mathcal{K}$, a question $\phi = \{f_1, \ldots, f_n\}$ is said to be **sound** if and only if for every fix $f_k = (A, i, t) \in \phi$ where $\Pi' = \Pi \cup \{(A, i)\}$, $\mathcal{K} = (\mathtt{apply}(\mathcal{F}, \{f\}), \Sigma_T, \Sigma_C)$ is Π'-repairable. An **inquiry** over $\mathcal{K}$ is a finite sequence of pair of sound questions and answers $\Omega_{\mathcal{K}} = ((\phi_1, f_1), \ldots, (\phi_n, f_n))$ such that $f_i \in \phi_i$.

A question ϕ is a set of fixes, whereas an answer is a fix that the user chooses from ϕ. In the framework, questions are sound if, once answered, will not render the knowledge base unrepairable. These questions are crucial to guide the user.[3]

Example 4.2. Consider the knowledge base of Example 1.1 and the following sound question:

[3]Hereafter, every question is meant to be sound.

- $\phi = \{(A, 1, X_1), (A, 2, X_2), (A', 1, Mike), (A', 1, X_3), (A', 2, Penicillin), (A', 2, X_4)\}$ such that:
  - $A = prescribed(Aspirin, John)$ and,
  - $A' = hasAllergy(John, Aspirin)$.

An inquiry is a sequence of tuples of question and answer. In what follows we show how a sound question can be generated and how an inquiry with a user takes place.

Algorithm 2 generates a sound question from a given conflict $\mathcal{X}$. The choice of a conflict being the starting point of a question is evident. In fact, fixing those atoms that are parts of some conflicts necessarily solves inconsistencies. The algorithm in line 4 generates all positions of the atoms of the conflict $\mathcal{X}$, then for each position (A, i) that does not belong to Π, we generate all possible fixes in lines 6-7. The fixes change the value of the position (A, i) to other values in the active domain different than the actual value and to an existential variable uniquely attributed to (A, i). Next in line 10, we enter in a filtering step where each fix is omitted if it renders the knowledge base not Π-repairable. Then it returns just ϕ. The following lemma proves that Algorithm 2 always gives a non-empty question which is necessarily sound.

LEMMA 4.3. *Given an inconsistent knowledge base $\mathcal{K}$ and a set of positions Π such that $\mathcal{K}$ is Π-repairable. Given a conflict $\mathcal{X} = (N, h)$, then* soundquestion$(\mathcal{K}, \Pi, \mathcal{X}) \neq \emptyset$ *and* soundquestion $(\mathcal{K}, \Pi, \mathcal{X})$ *outputs a sound question.*

PROOF. First, if $\Pi = \mathtt{pos}(\mathcal{F})$ then $\mathcal{K}$ is consistent (Π-repairability reduces down to consistency in this case), therefore there will be no conflict $\mathcal{X}$ in $\mathcal{K}$. Assume that $\Pi \subset \mathtt{pos}(\mathcal{F})$, then SOUNDQUESTION$(\mathcal{K}, \Pi, \mathcal{X}) = \emptyset$ if and only if: (1) In Line 5, $\Pi' \subseteq \Pi$, or, (2) In Line 8 , $val = \emptyset$ for each position $(A, i) \in P$, or, (3) In Line 16, every fix is removed from ϕ.

For (1), suppose it is the case. We know that $\mathcal{K}$ is Π-repairable, therefore there exists an r-fix $\mathcal{P}$ of $\mathcal{K}$ such that there exists no $(A, i, t) \in \mathcal{P}$ and $(A, i) \in \Pi$. Let $\mathcal{F}' = \mathtt{apply}(\mathcal{F}, \mathcal{P})$ be the update repair of $\mathcal{F}$ by $\mathcal{P}$. We know that $h(body(N)) \subseteq \mathcal{F}$, and $\forall A \in h(body(N))$ and for every $j \in [1, \mathtt{arity}(A)]$, $(A, i) \notin \mathcal{P}$ for some t because $(A, i) \in \Pi'$. Therefore, $h(body(N)) \subseteq \mathcal{F}'$, which means that $\mathcal{X} = (N, h)$ is a conflict in $\mathcal{K} = (\mathcal{F}', \Sigma_T, \Sigma_C)$, consequently $\mathcal{K}'$ is inconsistent and $\mathcal{F}'$ is not a u-repair, thus $\mathcal{P}$ is not r-fix, which contradicts the fact that $\mathcal{K}$ is Π-repairable. For (2), it cannot be the case because *val* can always hold $\{X^i_A\}$. For (3), it is clear at each iteration the fix $f_k = (A, i, X^i_A)$ is in ϕ and will not be removed because if $\mathcal{K}'$ is Π-repairable then $\mathcal{K}' = (\mathtt{apply}(\mathcal{F}, f_k), \Sigma_T, \Sigma_C)$ is also Π'-repairable where $\Pi' = \Pi \cup \{(A, i)\}$.

The fact that SOUNDQUESTION$(\mathcal{K}, \Pi, \mathcal{X})$ returns a sound question is quite straightforward since if SOUNDQUESTION$(\mathcal{K}, \Pi, \mathcal{X}) \neq \emptyset$, the Algorithm in line 14 drops any answer that does not lead to a Π-repairable knowledge base. □

This lemma tells us that if $\mathcal{K}$ is Π-repairable, we can always find a sound question. This relies on the intuition that $\mathcal{K}$ is Π-repairable, i.e. there necessarily exists some fixes that can be applied to render the knowledge base consistent.

Engaging the user in an inquiry needs to guarantee that the knowledge base is eventually repaired the way the user want it to be. However, such goal may never be accomplished if the inquiry cannot ensure that the resulting knowledge base is consistent. Algorithm 3 is the principled procedure that undertakes an inquiry dialogue with the user. The key idea is that we keep asking questions until there is no conflict left in the knowledge base. When the user chooses a fix (A, i, t) from ϕ, the position (A, i) becomes *immutable* to prevent modifying the value again. The algorithm terminates and produces a consistent knowledge base.

Algorithm 2 Generate sound question

```
1:  function SOUNDQUESTION(K, Π, X)
2:      X = (N, h)
3:      φ ← ∅
4:      Π' ← {(A, i) | A ∈ h(body(N)) and i ∈ [1, arity(A)]}
5:      for each (A, j) ∈ Π' \ Π do
6:          val ← adom(A, i, F) \ {value^i_A(F)}
7:          val ← val ∪ {X^i_A}
8:          φ ← φ ∪ {(A, i, t) | t ∈ val}
9:      end for
10:     for each f_k = (A, i, t) ∈ φ do
11:         Π_tmp ← Π ∪ {(A, i)}
12:         K' ← (apply(F, f), Σ_T, Σ_C)
13:         if Π-REP(K', Π'_tmp) = false then
14:             φ ← φ \ (A, i, t)
15:         end if
16:     end for
        return φ
17: end function
```

Algorithm 3 Inquiry with a user

```
1:  function INQUIRY(K, Π)
2:      K' ← K
3:      Π' ← Π
4:      while allconflicts(K') ≠ ∅ do
5:          Pick a conflict X ∈ allconflicts(K')
6:          φ ← SOUNDQUESTION(K, Π', X)
7:          f ← ASKUSER(φ)
8:          K' ← (apply(F, f), Σ_T, Σ_C)
9:          Π' ← Π' ∪ {(A, i) | f = (A, i, t)}
10:         Recompute allconflicts(K')
11:     end while
        return K'
12: end function
```

PROPOSITION 4.4 (SOUNDNESS AND TERMINATION). *Given an inconsistent knowledge base $\mathcal{K}$ and a set of positions Π. Then,* inquiry$(\mathcal{K}, \Pi)$ *returns a consistent KB $\mathcal{K}$ in a finite time.*

PROOF. Let $\mathcal{K}'_i$, Π'_i, ϕ_i, f_i be the knowledge base $\mathcal{K}'$, the set of positions Π', the sound question ϕ and the chosen fix f at the beginning of the while loop at round i.
Round 1: $\mathcal{K}'_1 = \mathcal{K}$ is inconsistent and Π'_1-repairable such that $\Pi'_1 = \Pi$.
Round i: $\mathcal{K}'_i$ is either consistent, therefore $\mathtt{allconflicts}(\mathcal{K}') = \emptyset$ and Algorithm 3 terminates, or inconsistent. However, we know that it is Π_i-repairable because $\mathcal{F}'_i = \mathtt{apply}(\mathcal{F}'_{i-1}, \{f_i\})$ such that $f_i \in \phi_i$ and ϕ_i is a sound question. Let this round be the one in which $|\mathtt{pos}(\mathcal{F}'_i)| - |\Pi'_i| = 1$. This means that at line 6 SOUNDQUESTION$(\mathcal{K}, \Pi, \mathcal{X}) \neq \emptyset$ and when the user chooses a fix in line 7, it is clear that at line 11 $\mathcal{K}'_i$ is Π'_i-repairable with $\Pi'_i = \mathtt{pos}(\mathcal{F}'_i)$. It is obvious that $\mathcal{K}'_i$ is consistent hence $\mathtt{allconflicts}(\mathcal{K}') = \emptyset$. □

In what follows, we investigate the complexity of Algorithm 2.

PROPOSITION 4.5. *In the worst-case, Algorithm 2 runs in $O(d \times (|\mathtt{pos}(\mathcal{F})| + C_{\Pi-rep}))$ with d being the size of the largest active domain in $\mathcal{K}$ and $C_{\Pi-rep}$ being the worst-case complexity of Π-repairability algorithm.*

PROOF. The worst-case corresponds to $\Pi = \emptyset$ and $h(body(N)) = \mathcal{F}$, i.e. the whole set of facts is a conflict. In this case, the loop at line 5 will iterate over all positions, i.e. $\mathtt{pos}(\mathcal{F})$. The loop at line 8 depends on d. The additional loop at line 10 performs d iterations.

We assume that the instruction at line 12 runs in constant time. Then, the function Π-REP$(\mathcal{K}', \Pi'_{tmp})$ is called d times. □

The ultimate and most desirable goal of the inquiry is to arrive at the user's repair. A well-founded framework is the one that meets such requirement. However, this depends on how the user answers the questions, his background knowledge and so on. Therefore, some assumptions have to be made. In the next section, we consider a special case, i.e. when the user is *an oracle*.

4.1 The Oracle

In this section, we discuss the case in which interaction takes place with an oracle O. The oracle corresponds to a u-repair $\mathcal{F}_O$ with an associated answering mechanism. The oracle draws its answers from $\mathcal{F}_O$ as follows: given a question ϕ, f_i is an oracle answer if and only if $f_i \in \texttt{diff}(\mathcal{F}, \mathcal{F}_O)$. In case of multiple answers from the oracle, O non-deterministically chooses one of them. Note that not all the sets of fixes in $\texttt{diff}(\mathcal{F}, \mathcal{F}_O)$ are necessarily r-fixes. There may exist finitely many set of fixes, even though we assume that if a given r-fix $\mathcal{P}_O$ is chosen by O, we name it an oracle r-fix. Note that such r-fix always exists as shown hereafter.

PROPOSITION 4.6. *Let $\mathcal{F}'$ be a u-repair of $\mathcal{F}$. Then, there exists a one-to-one correspondence match(x) such that* $\texttt{diff}(\mathcal{F}, \mathcal{F}')$ *is an r-fix.*

The proof is straightforward as there may exist exponentially many $match(x)$ that make all possible one-to-one correspondences. One of them must necessarily correspond to the real match because $|\mathcal{F}| = |\mathcal{F}'|$ and $\mathcal{F}'$ is homomorphic to $\mathcal{F}$.

It turns out that when interacting with the oracle, the oracle is capable of answering every question asked by Algorithm 3.

LEMMA 4.7. *Given a consistent knowledge base $\mathcal{K}$, a possibly empty set of positions Π, an oracle O and its chosen r-fix $\mathcal{P}_O$. Every question ϕ_i generated in* inquiry$(\mathcal{K}, \Pi)$ *contains at least a fix f_i such that f_i is in $\mathcal{P}_O$.*

PROOF. If there exists a sound question ϕ_i generated by INQUIRY$(\mathcal{K}, \Pi)$ at an iteration i such that $\phi_i \cap \mathcal{P}_O = \emptyset$ then there exists $f \in \mathcal{P}_O$ such that $\mathcal{K} = (\texttt{apply}(\mathcal{F}, \{f\}), \Sigma_T, \Sigma_C)$ is not Π'_i-repairable. Therefore, $\mathcal{K}$ has no u-repair. This contradicts the fact that $\mathcal{F}_O$ is a u-repair.

□

This lemma gives us the most important result, by stating that when the inquiry ends, the resulting knowledge base is in fact the oracle's u-repair $\mathcal{F}_O$.

PROPOSITION 4.8 (SOUNDNESS W.R.T O). *Let $\mathcal{K}' = (\mathcal{F}', \Sigma_T, \Sigma_C)$ be the knowledge base returned by* inquiry$(\mathcal{K}, \Pi)$ *with an oracle O as the user. Then, $\mathcal{F}'$ is the oracle's repair $\mathcal{F}_O$.*

PROOF. Since every question ϕ_i contains at least a fix $f \in \mathcal{P}_O$ then O will definitely choose a fix $f \in \mathcal{P}_O$. After answering by f, every next question ϕ_{i+1} will not contain f because once a fix is applied it will never be proposed again. However, by Lemma 4.7, ϕ_{i+1} will definitely contain a fix f' such that $f' \in \mathcal{P}_O \setminus \{f\}$. Hence, O will choose it until choosing all fixes in $\mathcal{P}_O$. We can see that in fact we are applying $\mathcal{P}_O$ on $\mathcal{F}$ one fix at a time. We know that $\mathcal{P}_O$ is an r-fix, thus in other words we are constructing a u-repair identical to $\mathcal{F}_O$. Therefore, $\mathcal{F}' = \mathcal{F}_O$.

□

Let us give an example of an inquiry with an oracle.

Example 4.9 (Inquiry with oracle). Consider the knowledge base of Figure 1 (b) and the oracle repair $\mathcal{F}_O$:

$$\mathcal{F}_O = \begin{cases} prescribed(Aspirin, John) \quad hasAllergy(Mike, Aspirin) \\ hasAllergy(Mike, Penicillin) \quad hasPain(Mike, Migraine) \\ isPainKillerFor(Nsaids, Migraine) \\ incompatible(Aspirin, Nsaids) \end{cases}$$

The inquiry is as follows:

(1) KB: which fix is true from the following set?
$\{(prescribed(Aspirin, John), 1, t) \mid t \in \{X_1, Nsaids\}\} \cup$
$\{(prescribed(Aspirin, John), 2, t) \mid t \in \{X_2, Mike\}\} \cup$
$\{(hasAllergy(John, Aspirin), 1, t) \mid t \in \{X_3, Mike\}\} \cup$
$\{(hasAllergy(John, Aspirin), 2, t) \mid t \in \{X_4, Penicillin\}\}$
(2) O: the fix $(hasAllergy(John, Aspirin), 1, Mike)$ is true.
(3) KB: which fix is true from the following set?
$\{(incompatible(Aspirin, Nsaids)), 1, X_5)\} \cup$
$\{(incompatible(Aspirin, Nsaids)), 2, X_6)\} \cup$
$\{(prescribed(Aspirin, John), 1, t) \mid t \in \{X_7, Nsaids\}\} \cup$
$\{(prescribed(Aspirin, John), 2, X_8)\} \cup$
$\{(hasPain(John, Migraine), 1, X_9)\} \cup$
$\{(hasPain(John, Migraine), 2, X_{10})\} \cup$
$\{(isPainKillerFor(Nsaids, Migraine), 1, X_{11})\} \cup$
$\{(isPainKillerFor(Nsaids, Migraine), 2, X_{12})\}$
(4) O: the fix $(hasPain(John, Migraine), 1, Mike)$ is true.

The knowledge base asks a question on a possible set of fixes. Then, the oracle chooses among them a fix that belongs to its r-fix. As one can notice after applying the fixes provided by the oracle in 2 and 4, the resulting knowledge base is indeed consistent and its set of facts equals $\mathcal{F}_O$. Notice that for instance the fix $f = (incompatible(Aspirin, Nsaids)), 1, X_5)$ has no proposed value other than the existential variable because the active domain is empty. An additional comment is in order. The size of the questions grows polynomially (and not exponentially) in the size of the conflicts and in the size of the active domain. As a consequence, presenting these questions to the user is an implementation concern that can benefit from advanced HCI techniques [8] and is beyond the scope of this paper.

When interacting with an oracle, Algorithm 1 performs as many iterations as the number of conflicts in the knowledge base. However, the number of iterations corresponds to the size of the oracle's r-fix, denoted as $\mathsf{r}_O^{\mathsf{num}}$. This is the case because the oracle at each step answers with a fix $f \in \mathcal{P}_O$ until all fixes in $\mathcal{P}_O$ are used. Given a set of empty positions Π and a Π-repairable inconsistent knowledge base $\mathcal{K} = (\mathcal{F}, \Sigma_T, \Sigma_C)$, then $\mathsf{r}_O^{\mathsf{num}} \leq |\mathsf{pos}(\mathcal{F})|$. Therefore, the number of iterations is linear in the size of $\mathcal{F}$. However, the algorithm is dominated by the complexity of SOUNDQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$ at line 6 and the computation of all conflicts. In fact, a conflict is the result of evaluating the body of a CDD, thus leading to a polynomial data complexity of boolean conjunctive query (for weakly-acyclic TGDs).

PROPOSITION 4.10. *Let C_{query} be the data complexity of evaluating a conjunctive query on a set of facts $\mathcal{F}$, in presence of a set of weakly-acyclic TGDs Σ_T, C_{soundq} be the complexity of* soundquestion *(Proposition 4.5) then the complexity of Algorithm 1 is $O(|\mathsf{pos}(\mathcal{F})| \times (|\Sigma_C| \times C_{query} + C_{soundq}))$.*

The above result gives us a polynomial delay algorithm.

COROLLARY 4.11. *Algorithm 3 takes a polynomial delay between questions.*

A polynomial delay algorithm is an algorithm in which the time between the output of the solutions is bounded by a polynomial function of the input size in the worst case [23]. In between questions, we perform query evaluation which costs $|\Sigma_C| \times C_{query}$ and the computation of a sound question which costs C_{soundq}. Therefore, the user will not have to wait from one question to the next more than an amount of time that is polynomially bounded.

5 QUESTIONING STRATEGIES

The goal of a strategy is to minimize the number of questions to be asked to the user in order to arrive at a consistent knowledge base. In this section, we present four strategies improving one on another. We introduce: the baseline strategy called *random*; another strategy, called *opti-join*, that improves over random by considering the so-called join positions; another variant of opti-join called *opti-prop* that uses propagation, and finally a fourth strategy, called *opti-mcd* that improves over *opti-join*.

First, let us define the lower and upper bounds of the number of questions for each strategy. It is obvious that the maximum number of questions is equal to $|\mathtt{pos}(\mathcal{F})|$. This case corresponds to a knowledge base in which every position needs to be changed to recover consistency. The minimum number of questions is clearly zero if the knowledge base is consistent.

The functions Π-REP$(\mathcal{K}, \Pi)$ of Π-repairability and CHECKCONSISTENCY$(\mathcal{K}')$ in Algorithm 1 and *recompute* $\mathtt{allconflicts}(\mathcal{K}')$ in Algorithm 3 shown in Section 4 are used in all the strategies. In the following, we detail Algorithm 4 where we propose an optimized version of these functions.

CHECKCONSISTENCY-OPT$(\mathcal{K}')$: the most naive approach for consistency check is to compute the chase on $\mathcal{F}$ to get $C\ell_{\Sigma_T}(\mathcal{F})$ then to check whether there exists a CDD whose body evaluates to true in $C\ell_{\Sigma_T}(\mathcal{F})$, as implemented in CHECKCONSISTENCY$(\mathcal{K}')$. The optimized version CHECKCONSISTENCY-OPT$(\mathcal{K}')$ considers CDDs and TGDs such that $\bot$ is seen as unary predicate (i.e. a constant). If, during the chase, the constant $\bot$ is produced then the knowledge base is inconsistent. This is quite fruitful as it helps to stop consistency check as early as possible.

Π-REPOPT$(\mathcal{K}, \Pi)$: we can easily observe that if a knowledge base $\mathcal{K}$ is Π-repairable and some positions have been fixed using a set of values $\mathcal{V}$ then in the case in which a new fix f arrives with value v (constant or fresh existential variable), the knowledge base stays Π-repairable if $v \notin \mathcal{V}$. This is quite intuitive because if the fixed positions do not trigger any CDDs, the new value will not trigger any CDDs since all atoms have different values. If the value is already used, we proceed to the optimized consistency check CHECKCONSISTENCY-OPT$(\mathcal{K}')$. This is the optimized Π-repairability check of Π-REP$(\mathcal{K}, \Pi)$.

Let us now turn to the optimization of $\mathtt{allconflicts}(\mathcal{K}')$. Let $\mathtt{allconflicts}_{\mathtt{naive}}(\mathcal{K}')$ be defined as the set of all naive conflicts. A naive conflict $\mathcal{X} = (N, h)$ is defined as a conflict in the sense of Definition 2.3 except that h is a homomorphism from $body(N)$ to $\mathcal{F}$ such that $h(body(N)) \subseteq \mathcal{F}$ and $C\ell_{\Sigma_T}(\mathcal{F}) \models \bot$. These conflicts are computed on $\mathcal{F}$ without applying the chase. It is clear that if $\mathtt{allconflicts}_{\mathtt{naive}}(\mathcal{K}') = \emptyset$, $\mathcal{K}$ is not necessarily consistent as there is the possibility of having conflicts that will appear after applying the chase like the conflict $\mathcal{X}_2$ in Example 2.4. However, we observed that resolving naive conflicts at first can eliminate other conflicts that are discovered after applying the chase. For instance in Example 2.4, if we resolve the conflict $\mathcal{X}_2$ by updating the atom $prescribed(Aspirin, John)$ on the first position, this will resolve the conflict that can be detected using the second CDD after applying the TGD. Therefore, our strategies are *two-phases* strategies. In the first phase, naive conflicts are resolved, while in the second phase, if the KB is still inconsistent, new conflicts are discovered and resolved during the chase. In what follows, we provide an optimization of conflicts computation.

UPDATECONFLICTS$(\mathcal{K}')$: we compute the initial set of naive conflicts over $\mathcal{K}$ and keep them in a set C_{naive}, then C_{naive} is updated as follows. If the user provides a fix $f = (A, i, t)$ which results in a new set of facts $\mathcal{F}'$, then we remove all conflicts that are related to A from C_{naive}. Next, we define a subset $\Sigma_C^A \subseteq \Sigma_C$ that is related to A as follows: a CDD $N \in \Sigma_C^A$ iff $\exists A' \in body(N)$ and a homomorphism h such that $h(A') = A$. Finally, C_{naive} is updated by evaluating the body of each CDD $N \in \Sigma_C^A$ over the new set of facts $\mathcal{F}'$. In this optimization, instead of recomputing all conflicts, we are limiting the computation to the modified atom which is more efficient than evaluating all CDDs on $\mathcal{F}'$.

Once we have defined the above optimizations, we turn our attention to our proposed strategies, namely *random*, *opti-join*, *opti-prop* and *opti-mcd*. Algorithms 4 & 5 are parametrized, thus we can easily plug in the above optimizations. The main code of each strategy is Algorithm 4 which calls the functions GENERATEQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$ and GENERATEQUESTION-CHASE$(\mathcal{K}, \Pi', \mathcal{X})$. These functions are implemented differently for each strategy. In addition, these functions make use of SOUNDQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$ for which the function RETRIEVE-POSITIONS$(\mathcal{X}, \mathcal{K})$ changes from a strategy to another. For space reasons, in the following we report a concise description of each strategy. Their implementation and effectiveness are discussed in the next section.

Random. This strategy selects randomly a conflict from C_{naive} before asking a question about all positions. More precisely, GENERATEQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$ randomly picks a conflict from C_{naive} and calls SOUNDQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$. Then, RETRIEVE-POSITIONS$(\mathcal{X}, \mathcal{K})$ for each atom in $h(body(N))$, generates all positions (A, i) and proceed normally in SOUNDQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$. While applying the chase if a violation of a CDD is detected, GENERATEQUESTION-CHASE$(\mathcal{K}, \Pi', \mathcal{X})$ gets all facts in $\mathcal{F}$ that contribute to its violation, then generates all positions from this set and returns it as a question using SOUNDQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$.

Opti-join. This strategy improves over *random* on RETRIEVE-POSITIONS$(\mathcal{X}, \mathcal{K})$ where only join positions are generated. Given a conflict $\mathcal{X} = (N, h)$, a position (A, i) is a join position if and only if the variable at the position i in A' is a join variable in the CDD N such that $h(A') = A$. For instance, consider the example of Figure 1(b) the position $(prescribed(Aspirin, John), 1)$ is a join position because the variable X in $A' = prescribed(X, Y)$ is a join variable in the second CDD. Clearly, this strategy generates smaller questions and most notably avoid asking unnecessary questions. Consider the knowledge base $\mathcal{K}$ without TGDs:
$\mathcal{F} = \{isUrgent(Mike, a, 145), isDeferredTo(Mike, 12/10/2015)\}$.
$\Sigma_C = \{isUrgent(X, Y, Z), isDeferredTo(X, W) \rightarrow \bot\}$.

Here the position $(isDeferredTo(Mike, \text{“}12/10/2015\text{”}), 2)$ is not a join position. However, the position $(isUrgent(Mike, a, 145), 1)$ is a join position. Join positions are pivotal in order to resolve conflicts, since changing non-join positions does not affect the homomorphisms and does not resolve conflicts.

Opti-prop. This strategy behaves the same as *opti-join* except that a propagation technique is used. By definition if the user chooses a fix $f = (isUrgent(Mike, a, 145), 1, X_1)$ from a question ϕ produced from a conflict $\mathcal{X}$, then every position generated from $\mathcal{X}$ (except the chosen one in f) is added to Π if and only if it is not involved in any other conflict $\mathcal{X}'$. This is quite intuitive because when the user chooses a fix f from a question ϕ, he is implicitly indicating that they are non-erroneous. However, if some of these positions participate in other conflicts then it is possible that they are erroneous, thus they will not be added to Π.

Opti-mcd. This strategy is an improvement over *opti-join*, it is based on the so-called Conflict Hypergraph (CH) [13, 24] where GENERATEQUESTION$(\mathcal{K}, \Pi', \mathcal{X})$ and GENERATEQUESTION-CHA

SE($\mathcal{K}, \Pi', \mathcal{X}$) compute the **M**aximally **C**ontain**D** position in all computed conflicts. Each position p is attributed a rank that indicates the number of conflicts containing p. The question is generated on the position that has the maximum rank. If more than one position are attributed the same maximum rank, one is picked randomly. Using CH, the maximally contained position p corresponds to the vertex of maximum degree. Obviously, this strategy can avoid asking unnecessary questions by looking ahead and choosing the position that resolves as many conflicts as possible.

Algorithm 4 Inquiry strategy

```
1: function INQUIRY(K, Π)
2:     K' ← K
3:     Π' ← Π
4:     C_naive = allconflicts_naive(K')
5:     /* Start phase one */
6:     while C_naive ≠ ∅ do
7:         φ ← GENERATEQUESTION(K, Π', X)
8:         f ← ASKUSER(φ)
9:         K' ← (apply(F, f), Σ_T, Σ_C)
10:        Π' ← Π' ∪ {(A, i) | f = (A, i, t)}
11:        UPDATECONFLICTS(K')
12:    end while
13:    /* Start phase two */
14:    while CHECKCONSISTENCY-OPT(K') = false do
15:        φ ← GENERATEQUESTION-CHASE(K, Π', X)
16:        f ← ASKUSER(φ)
17:        K' ← (apply(F, f), Σ_T, Σ_C)
18:        Π' ← Π' ∪ {(A, i) | f = (A, i, t)}
19:    end while
       return K'
20: end function
```

Algorithm 5 Sound questions for a strategy

```
1: function SOUNDQUESTION(K, Π, X)
2:     X = (N, h)
3:     φ ← ∅
4:     Π' ← RETRIEVE-POSITIONS(X, K)
5:     for each (A, j) ∈ Π' \ Π do
6:         val ← adom(A, i, F) \ {value^i_A(F)}
7:         val ← val ∪ {X^i_A}
8:         φ ← φ ∪ {(A, i, t) | t ∈ val}
9:     end for
10:    for each f_k = (A, i, t) ∈ φ do
11:        Π_tmp ← Π ∪ {(A, i)}
12:        K' ← (apply(F, f), Σ_T, Σ_C)
13:        if Π-REPOPT(K', Π'_tmp) = false then
14:            φ ← φ \ (A, i, t)
15:        end if
16:    end for
       return φ
17: end function
```

6 EXPERIMENTAL STUDY

Our experimental assessment is devoted to study two major features of our user-guided repairing framework: (i) *effectiveness:* investigates to what extent the framework is efficient in helping the user repair the knowledge base with minimal effort, in terms of average number of asked questions per strategy and average number of conflicts per question, and (ii) *delay time:* the efficiency of our framework when it comes to maintaining a reasonable delay time between each asked question. By a reasonable delay time we intend a delay less than 1 to 2 seconds as discussed in [26].

Experimental setup. We have implemented our framework using Java 1.8 on a 2.40GHz 4-core, 16Gb laptop running Windows 7. We have used GRAAL[4] as a chase engine. Each experiment has been repeated a number of times, as indicated in the individual plots (after discarding the cold start). As there are no existing datasets or benchmarks equipped with the rich set of constraints we consider in this paper, we rely on synthetically generated knowledges bases and corresponding constraints. We also employ a real-world knowledge base on Durum Wheat from [2]. This knowledge base has been constructed manually from documents and reports, which led to have notable inconsistencies. Moreover, the attached constraints (including TGDs and CDDs) have been validated by experts. Such a KB turned to be suitable for our experiments as it fits the assumption that the set of facts is dirty and the set of constraints is reliable. [5]

Synthetic KBs. The synthetic knowledges bases were generated by tuning some input parameters. We first generate a *vocabulary* of the knowledge base, i.e. predicate, variable, and constant spaces by allowing also n-ary relations. Each predicate is assigned a random arity from 2 to 10 following a uniform probability distribution. A given number of CDDs are generated over the vocabulary by parameterizing the number of atoms s involved in the CDDs and the percentage $v_{\%}$ of atoms positions corresponding to join variables such that $s \in [5, 10]$ and $v_{\%} \in [10\%, 100\%]$. TGDs are generated following the same procedure as CDDs. To make the knowledge base meaningful, links between TGDs and CDDs are made so that some TGDs may introduce facts that will violate the CDDs. A *depth* $d_{\mathcal{K}}$ for a given knowledge base $\mathcal{K}$ is defined as how many TGDs applications are needed to violate a CDD. A conflict depth $d_{\mathcal{K}} = 2$ means for each CDD we need the application of two distinct TGDs so that the CDD is violated. Inconsistent KBs are generated as follows, for a given facts size $n_{\mathcal{F}}$ and an *inconsistency ratio* r_{inc} [6], we keep generating sets of atoms that violate the CDDs until we reach r_{inc}. Then, we pad the set of facts $\mathcal{F}$ with atoms that are not involved in any conflicts. We add two indicators of the structure of conflicts in the KB, namely "Avg # atoms per overlap" and "Avg scope". The first embodies the average number of atoms in each overlap, an overlap being the intersection between at least two conflicts. The avg scope indicates for each conflict how many conflicts are overlapping with it, this number being averaged over the total number of conflicts.

The Durum Wheat KBs. The real-world Durum Wheat knowledge base in our experiments has been augmented with new domain-specific TGDs and CDDs. The table in Figure 2 presents the different characteristic of the knowledge bases and an example of facts, a TGD and a CDD. Please note that ChaseSize (#atoms) refers to the size of the facts after applying the chase. We made two versions of the knowledge base of increasing size of CDDs, i.e. Durum Wheat v1 and Durum Wheat v2. Notice that the number of conflicts increases from v1 to v2 while inconsistency ratio stays the same. This is due to the fact that the conflicts newly discovered by the added constraints in v2 involve the same number of atoms.

We have simulated the end-user via an algorithm that randomly chooses a valid fix from the proposed fixes following a uniform probability distribution. Considering other kinds of distributions and, in particular, choosing the most appropriate probability distribution that can simulate all the user's choices is not trivial and falls under user modeling, which is beyond the scope of our paper.

[4] http://graphik-team.github.io/graal/.

[5] Notice that we could not use popular knowledge bases such as YAGO, DBPedia and LUBM because of their limited expressiveness on the vocabulary (only binary relations) and lack of TGDs and CDDs.

[6] Number of atoms involved in at least one conflict divided by $n_{\mathcal{F}}$.

Analysis of Durum Wheat KBs. We measure the average number of asked questions for each strategy in order to gauge the effectiveness of our approach on our real-world dataset. Figure 2 (a) and (b) show the results for all the considered strategies. We can observe that opti-mcd is outperforming the other strategies on Durum Wheat v1 with an average of 14.18 questions asked. This difference is also observed on Durum Wheat v2 in Figure (a) where opti-mcd outperforms other strategies with an average of 29.36 questions asked. The reason why opti-mcd is the winning strategy is due to the fact that this strategy is actually capable of exploiting the overlapping among conflicts which is given by the indicator avg scope. In this case, the value of such indicator is 8 meaning that in the best case, roughly speaking, each question can solve 8.1 (or 7.1 for V2) conflicts. This is under the assumption that the conflicts are overlapping on the same atoms. Regarding the difference with the other strategies, the results show that the strategies (other than opti-mcd) tend to behave the same as they do not exploit such a property. In addition, notice that opti-join and opti-prop are very close to random strategy. This is due to the fact that the percentage of join positions in conflicts is close to 90%. This makes the probability of choosing a join position with random strategy very high. Moreover, the increase in the average number of asked questions in all strategies in v2 is explained by the fact that v2 has slightly more conflicts than v1.

Another perspective that gives a better illustration of the effectiveness of our interactive strategies is the average number of resolved conflicts per question in Figures (c) and (d). The former is computed as total nr. of conflicts/total nr. of questions (per strategy). Again, we can observe that the opti-mcd strategy handles more conflicts per question on average and proves to be the most effective strategy compared to the others.

Analysis of Synthetic KBs. We now analyze the effectiveness where the average number of asked questions is measured for each strategy on synthetically generated KBs. For the first experiment described in Figure 3, we generated a knowledge base with only CDDs and no TGDs. Then, we increased the inconsistency ratio by increments of 5% while keeping the size of the knowledge base is fixed (see the table in Figure 3 for characteristics). The results show a good performance of opti-mcd, while opti-join and opti-prop behave similarly. The random strategy performs the worst among them. This result in fact confirms the observation already made on the Durum Wheat knowledge base about overlapping conflicts. In this experiment, we notice a larger gap between opti-join and opti-prop strategies on one side and random strategy on the other side. Since the percentage of join positions in conflicts is very low in the generated atoms (under to 30%), it is less likely that the random strategy would randomly choose a join position. The average number of resolved conflicts per question is shown in Figure 3 (b) where one can see the performance of each strategy.

We should highlight that the baseline strategy (*random*) in the two experiments is performing quite well as it still asks less questions than the number of conflicts. This is quite natural as the conflicts are in fact overlapping (as confirmed by the two indicators, avg # atoms per overlap and avg scope) hence many conflicts are resolved via the resolution of other conflicts. However, there is a huge gap between the performance of the baseline strategy and those of the more optimized strategies.

The second experiment aims at studying the convergence of each strategy. Figure 4 (a) is done on an inconsistent knowledge base with CDDs and no TGDs. We can observe that as the strategies proceed with questioning, they exhibit different speeds in getting toward a full resolution of the conflicts. While opti-mcd is faster than all the other strategies, opti-join and opti-prop are quite similar with a small difference on the number of asked questions. Figure 4 (b) is done on a fixed inconsistent knowledge base with both CDDs and TGDs. We can observe that each strategy hits the lowest number of conflicts at a point (close to 0) then it starts slightly fluctuating until convergence. The rapid descending phase corresponds to the process of handling only CDDs that are directly violated by the initial set of facts without taking into account the TGDs. Once the TGDs are triggered and the chase starts, it interleaves TGDs with CDDs, leading to up and down fluctuations corresponding to new conflicts introduced by TGDs and resolution of conflicts with CDDs, respectively. Continuous lines between fluctuations correspond to stagnation, in which neither the questions are resolving conflicts nor triggering TGDs and CDDs brings new conflicts. The strategies opti-mcd, opti-join and opti-prop behave quite similarly with a notable difference in the convergence speed, bringing opti-mcd to be the fastest.

The next experiment is devoted to measure the delay time between questions for synthetic KBs. Note that the delay time for all previous experiments was very reasonable (less than 0.2 seconds) for both synthetic and Durum Wheat knowledge bases.

Figure 5 shows three different measures of the delay time using opti-mcd in all experiments. The delay time for the other strategies had a similar trend and is omitted for space reasons.

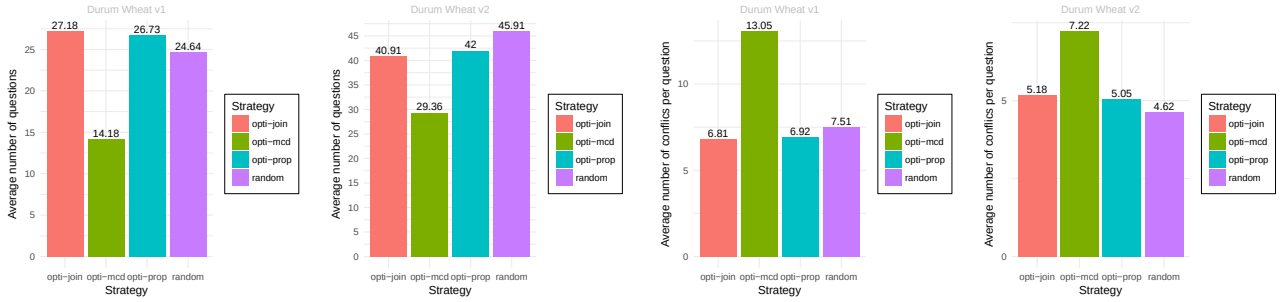
The goal of the experiment whose results are reported in Figure 5 (a) is to investigate whether increasing inconsistency (from 20% to 80%) would affect the delay time. We can observe that the inconsistency ratio is rather independent from the delay time. This is quite interesting as regardless of the inconsistency degree of the knowledge base, interactivity is guaranteed with the user in a very reasonable time (average is less than 0.25 sec). Some outliers are highlighted in the boxplot, however they stay within the limits of reasonable delay time (less than 0.8 sec).

In the next experiment (Figure 5 (b)), we employed a KB of increasing size (up to 20%, 40% and 60%), respectively while keeping the inconsistency ratio fixed to 30%. The delay time grows as the size of the knowledge base grows. Moreover, the boxplot shows that the variance of delay time increases as the size of the KB increases. This result shows that our method needs a piecemeal application of interactive repairing and can always be applied to small portions of the KB.

In the next experiment, we have chosen a worst-case scenario in which we have a fully inconsistent KB, corresponding to inconsistency ratio of 100%, and we vary the depth (from d_1 to d_4) of the dependencies involved (both TGDs and CDDs). For all depth d_i we have $\#CDD(d_i) = 150$, and $\#TGDs(d_1) = 50, \#TGDs(d_2) = 100, \#TGDs(d_3) = 150, \#TGDs(d_4) = 200$. We have already shown in the experiment of Figure 5 (a) that increasing the inconsistency ratio does not affect the delay time. We observe that the delay time increases with depth, in fact the larger the depth the more time the chase takes while repairing. Notice that the chase is involved in computing Π-repairability and consistency check. Overall, the delay time is kept low for all depths and less than 2 seconds.

7 CONCLUSION AND FUTURE WORK

In this paper, we have presented a novel user-guided repairing technique for knowledge bases, leveraging updates and interplay of dependencies (TGDs and CDDs). Several extensions can be thought of, such as formalization of user modeling to represent several classes of users (from domain experts to non-experts), and learning from provided user choices in the questioning strategies.

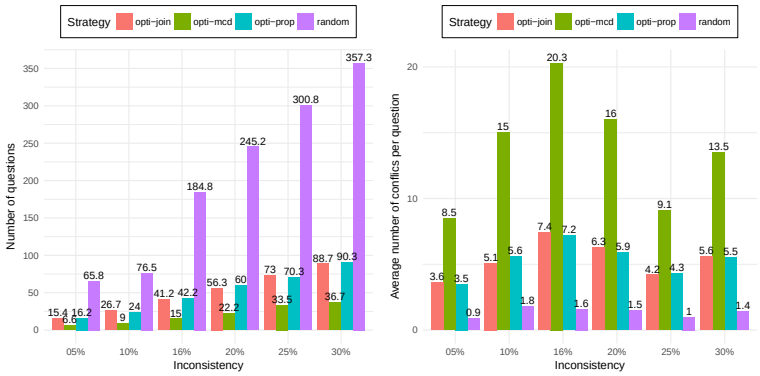


(a) (b) (c) (d)

KB	Size (#atoms)	ChaseSize (#atoms)	Conflicts	Avg # atoms per overlap	Avg scope	#Repetitions
DURUM WHEAT V1	567	1075	185	1.42	8.1	10
DURUM WHEAT V2	567	1075	212	1.41	7.8	10
KB	#TGDs	#CDDs	Inconsistency ratio	Avg # atoms per conflict		
DURUM WHEAT V1	269	27	14% (79 atoms)	3		
DURUM WHEAT V2	269	100	14% (79 atoms)	2		

Durum Wheat KB	Content
$\mathcal{F}$	$hasPrecedent(soil2, vacoparis), sorghum(vacoparis), soil(soil2).$ soil2 has a precedent vacoparis of type sorghum.
Σ_T	$isCultivatedOn(X1, X2), durum\_wheat(X1), soil(X2) \rightarrow hasPrecedent(X2, X3), soybean(X3)$ if a durum wheat is cultivated on a soil then the precedent on this soil is soybean.
Σ_C	$isAtGrowingStage(X, Z), isPerformedOn(X1, X), tillering\_begins(Z), durum\_wheat(X), fertilization(X1) \rightarrow \bot$ it is forbidden (or impossible) to apply a fertilization on a durum wheat if it is in the beginning of the tillering growth stage.

Figure 2: Average number of questions per strategy on the Durum Wheat knowledge bases.



(a) Fixed size KB (1005 atoms) with increasing inconsistency ratio. CDDs and no TGDs.

(b) Average conflicts per question of (a).

KB	Size (#atoms)	ChaseSize (#atoms)	Conflicts	Avg # atoms per overlap	Avg scope	#Repetitions
"05%"	1005	1005	56	1.45	9.8	6
"10%"	1005	1005	135	1.57	12.8	6
"16%"	1005	1005	304	1.68	33.9	6
"20%"	1005	1005	356	1.65	30.5	6
"25%"	1005	1005	304	1.67	34.5	6
"30%"	1005	1005	496	1.66	31.6	6

Figure 3: Average number of questions per strategy on synthetic knowledge bases. The percentage on the axes represent inconsistency ratio.

We also believe that more challenges may arise in extending this work to full-fledged denial constraints and arbitrary (non weakly acyclic) TGDs.

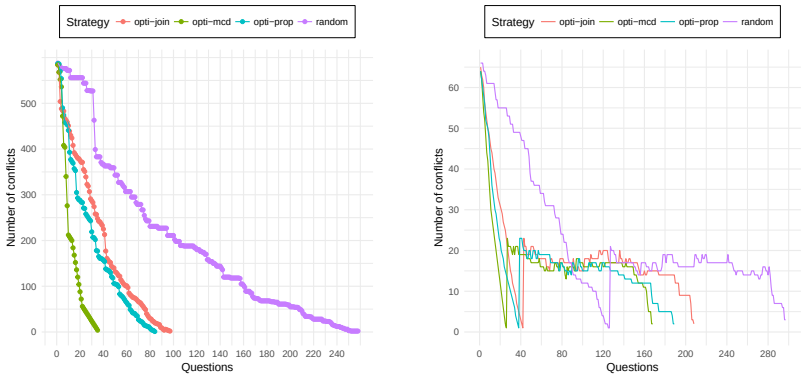
REFERENCES

[1] Marcelo Arenas, Leopoldo Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proc of SIGMOD*. ACM, 68–79.

[2] Abdallah Arioua, Patrice Buche, and Madalina Croitoru. 2016. A Datalog ± Domain-Specific Durum Wheat Knowledge Base. In *Proc. of MTSR 2016*. Springer, 132–143.

[3] Sebastian Arming, Reinhard Pichler, and Emanuel Sallinger. 2016. Complexity of Repair Checking and Consistent Query Answering. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 48. 21:1–21:18.

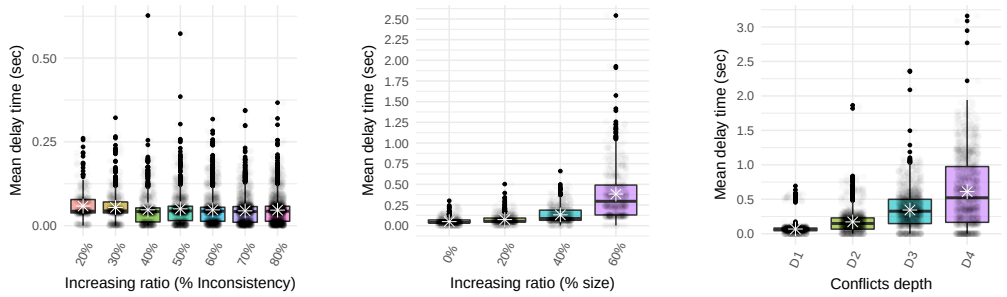
[4] Ahmad Assadi, Tova Milo, and Slava Novgorodov. 2017. DANCE: Data Cleaning with Constraints and Experts. In *Proc. of ICDE*. 1409–1410.



(a) Fixed size KB (3004 atoms) with constant inconsistency ratio 25%. With only CDDs and no TGDs.

(b) Fixed size KB (800 atoms) with constant inconsistency ratio of 25%, 50 CDDs and 25 TGDs. Total number of conflicts after applying the chase is 136.

Figure 4: The convergence of strategies over a question/answer session.



(a) Fixed size KB (3000 atoms) with increasing inconsistency ratio.

(b) Increasing size KB starting at 0% with 3000 atoms, constant inconsistency ratio 30%.

(c) Fixed size KB (400 atoms) with constant inconsistency ratio of 100%, variated depth (1 to 4).

Figure 5: Average delay time with 5 repetitions for each percentage. (c) is the delay time when TGDs and CDDs are considered, $\#CDD(d_i) = 150$, and $\#TGDs(d_1) = 50, \#TGDs(d_2) = 100, \#TGDs(d_3) = 150, \#TGDs(d_4) = 200$. *Opti-mcd* strategy is used. Asterix represents the mean.

[5] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175, 9-10 (2011), 1620 – 1654.

[6] Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. 2017. Benchmarking the Chase. In *Proc. of PODS*. 37–52.

[7] Leopoldo Bertossi. 2011. Database repairing and consistent query answering. *Synthesis Lectures on Data Management* 3, 5 (2011), 1–121.

[8] Sourav S. Bhowmick, Byron Choi, and Chengkai Li. 2017. Graph Querying Meets HCI: State of the Art and Future Directions. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. 1731–1736.

[9] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional Functional Dependencies for Data Cleaning. In *Proc. of ICDE*. 746–755.

[10] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proc. of SIGMOD*. 143–154.

[11] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2012. A general Datalog-based framework for tractable query answering over ontologies. *Journal of Web Semantics* 14 (2012), 57–83.

[12] Andrea Cali, Georg Gottlob, and Andreas Pieris. 2012. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence* 193 (2012), 87–128.

[13] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *Proc. of ICDE*. IEEE, 458–469.

[14] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *Proc. of SIGMOD*. 1247–1261.

[15] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.

[16] Wenfei Fan and Ping Lu. 2017. Dependencies for Graphs. In *Proc. of PODS*. 403–416.

[17] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *Proc. of SIGMOD*. 1843–1857.

[18] Mina H. Farid, Alexandra Roatis, Ihab F. Ilyas, Hella-Franziska Hoffmann, and Xu Chu. 2016. CLAMS: Bringing Quality to Data Lakes. In *Proc. of SIGMOD*. 2089–2092.

[19] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC Data-Cleaning Framework. *PVLDB* 6, 9 (2013), 625–636.

[20] Víctor Gutiérrez-Basulto, Yazmín Ibáñez García, Roman Kontchakov, and Egor V. Kostylev. 2015. Queries with Negation and Inequalities over Lightweight Ontologies. *Web Semant.* 35, P4 (Dec. 2015), 184–202.

[21] Jian He, Enzo Veltri, Donatello Santoro, Guoliang Li, Giansalvatore Mecca, Paolo Papotti, and Nan Tang. 2016. Interactive and Deterministic Data Cleaning. In *Proc. of SIGMOD*. 893–907.

[22] Neil Immerman. 1989. Expressibility and parallel complexity. *SIAM J. Comput.* 18, 3 (1989), 625–638.

[23] David S. Johnson, Mihalis Yannakakis, and Christos H. Papadimitriou. 1988. On generating all maximal independent sets. *Inform. Process. Lett.* 27, 3 (1988), 119 – 123.

[24] Solmaz Kolahi and Laks VS Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*. ACM, 53–62.

[25] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. 2010. Inconsistency-tolerant Semantics for Description Logics. In *Proceedings of the International Conference on Web Reasoning and Rule Systems (RR'10)*. Springer-Verlag, 103–117.

[26] Robert B. Miller. 1968. Response Time in Man-computer Conversational Transactions. In *Proc. of FJCC 1968*. ACM, 267–277.

[27] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J. Miller, and Divesh Srivastava. 2015. Combining Quantitative and Logical Data Cleaning. *PVLDB* 9, 4 (2015), 300–311.

[28] Jef Wijsen. 2005. Database repairing using updates. *ACM TODS* 30, 3 (2005), 722–768.

[29] Mohamed Yakout, Ahmed K. Elmagarmid, Jennifer Neville, Mourad Ouzzani, and Ihab F. Ilyas. 2011. Guided data repair. *PVLDB* 4, 5 (2011), 279–289.